UNIVERSIDADE DE ÉVORA

5º Semestre
Ano Letivo 2020/2021

Inteligência Artificial

2º Trabalho Prático -

Resolução de problemas como problemas de satisfação de restrições

	1				8		7	3
			5		9			
7						9		4
					4			
				3	5		1	8
8			9					
			7					
2	6			4			3	
		5			3			

Trabalho realizado por:

Pedro Grilo, 43012

Diogo Castanho, 42496

Rodrigo Vicente, 42844

Introdução

Com a resolução deste trabalho é pretendido que sejam utilizadas as capacidades de resoluções de problemas como problemas de satisfação de restrições.

Para isso, deveremos conseguir perceber as diferenças entre as pesquisas backtracking e a backtracking com forwarding, percebendo também as variáveis existentes, como o nome, domínio e valor, e as restrições para cada exercício.

Resolução dos Exercícios Propostos

1º Exercício

a) Para representar os estados, decidimos as variáveis apenas com o sitio da cadeira onde vão estar v(c(Numero_cadeira),D,Valor), e o valro das mesmas.

O domínio será o nome de todos as pessoas que podem estar sentadas nas cadeiras - **dominio**([’Maria’, ’Manuel’, ’Madalena’, ’Joaquim’, ’Ana’, ’Julio’, ’Matilde’, ’Gabriel’].

Nas restrições fizemos um predicado restrições, com as restrições a que devemos obedecer (esquerda, frente, lado e direita, como está no enunciado).
Depois no predicado restric, vamos descobrir todas as possibilidades usando as restrições das pessoas que se podem ou não sentar nas cadeiras.

Seguem-se abaixo mesmas **restrições** explicadas acima:

```
ve_restricoes(e(_Nafec,Afect)):- \+ (member(v(c(I),_Di,Vi), Afect),
                                    member(v(c(J),_Dj,Vj),Afect),
                                    I \=J,  %sucede se violar a restrição
                                    restric(I,Vi,J,Vj)).   %VI = VJ vê se há rainhas na mesma casa

%restrict sucede se alguma restrição falha
restric(I,X,J,Y) :- restricoes(L), member(esq(X,Y),L), \+ (I is J+1; (I=1, J=8)).   %restrições(L) é a lista com as restri

restric(I,X,J,Y) :- restricoes(L), member(dir(X,Y),L), \+ (I is J-1; (I=8, J=1)).

restric(I,X,J,Y) :- restricoes(L), member(lado(X,Y),L), \+ ((I is J-1; (I=1, J=8)); (I is J+1; (I=1,J=8))).

restric(I,X,_,_) :- restricoes(L), member(cabeceira(X),L), \+ (I=5, I=1).

restricoes([esq('Manuel','Maria'), frente('Joaquim','Maria'),lado('Joaquim','Matilde'),cabeceira('Gabriel')]).
```

O nosso **estado inicial**, com o domínio referido:

```
estado_inicial(e([
    v(c(1),D,_),                    %cadeira 1....
            v(c(2),D,_),            %cadeira 2....
                        v(c(3),D,_),
            v(c(4),D,_),
            v(c(5),D,_),
                        v(c(6),D,_),
            v(c(7),D,_),
            v(c(8),D,_)],[])) :- D = ['Maria', 'Manuel', 'Madalena', 'Joaquim', 'Ana', 'Julio','Matilde', 'Gabriel'].
```

Já o **operador sucessor** é o seguinte:

```
sucessor(e([v(N,D,V)|R],E),e(R,[v(N,D,V)|E])):- member(V,D).
```

b) Problema resolvido com algoritmo de backtracking:
Usando este algoritmo obtemos um conjunto infinito de soluções para a mesa, provavelmente devido à forma de escrita que aplicámos aqui, mas não conseguimos fazer de outra maneira. Para além disso, pode também ser por algum erro nas restrições dadas.

c) Problema resolvido com algoritmo de backtracking com pesquisa forward:

Por alguma razão, a nossa pesquisa forward só conseguiu chegar ao resultado final da mesa com 4 pessoas e com 6 pessoas, pelo que para o resto dos exemplos não deu nenhuma solução (apesar de escrever uma grande quantidade de bytes, a maioria mesas repetidas, com nomes repetidos).

d) Para este problema não percebemos como deveríamos proceder para aumentar para ainda melhor a complexidade de resolução do problema.

e) A. 4 pessoas na mesa (1,2,3,4 as cadeiras disponíveis para as 8 pessoas)

Com pesquisa back (mais demorada)

```
c(4) - Manuel
c(3) - Ana
c(2) - Madalena
c(1) - Matilde
```

Com pesquisa forward (mais rápida)

```
c(4) - Madalena
c(3) - Ana
c(2) - Julio
c(1) - Matilde
```

B. 6 pessoas na mesa

Com pesquisa back não obtivemos resultados corretos (apenas mesas com nomes repetidos)

Com pesquisa forward encontrámos alguns exemplos, como este:

```
c(6) - Ana
c(5) - Manuel
c(4) - Maria
c(3) - Madalena
c(2) - Joaquim
c(1) - Julio
```

C. 8 pessoas na mesa

Com pesquisa back obtivemos resultados, mas um loop infinito dos mesmos, a maioria repetidos.

Com pesquisa forward não foram encontrados quaisquer resultados.

D. 12 pessoas na mesa

Com pesquisa back obtivemos resultados, mas um loop infinito dos mesmos, a maioria repetidos.

Com pesquisa forward não foram encontrados quaisquer resultados.

Pelo que, não obtivemos grandes valores para aquilo que deveríamos ter obtido, apesar de percebermos o pedido e achando que as restrições estão corretas.

Instruções para compilar o problema da zebra:

1. Abrir o terminal na pasta com os ficheiros
2. Abrir prolog
3. Compilar [pesquisaback] para pesquisa backtracking, ou [pesquisaforward] para pesquisa backtracking com forward checking.
4. Para além disso, dentro de ambas as pesquisas temos de fazer consult(Zebra) (dentro do código das pesquisas), alterando as pessoas para aquelas que queremos ter na mesa.
5. Posteriormente, apenas temos que escrever "p." no terminal e irão ser escritas as soluções das mesas.

2º Exercício

f) Para representar os estados, decidimos ter variáveis com um tuplo com 3 números (X,Y,Z). O X e o Y representam as coordenadas do quadrado da tabela do sudoku, e o terceiro, Z, representa o quadrante onde está inserido (sendo que o tabuleiro do sudoku tem 9 quadrantes (3x3)).
O domínio vai de [1 a 9], uma vez que temos 3x3 quadrantes.
Para o valor, apenas dizemos o número relativo ao mesmo (se já existir no sudoku inicial, ou "_", no caso de não existir).

Nas restrições, decidimos através de findalls encontrar todos os valores existentes para todas as posições com um mesmo X (que se encontram na mesma linha), e através de um predicado all_diff ver se todos os valores encontrados são diferentes. Repetimos o mesmo para Y e o Z, ou seja, para as colunas e os quadrantes do tabuleiro respetivamente.
Seguem-se as mesmas **restrições** explicadas acima:

```
ve_restricoes(e(_,[v(c(X,Y,Z), _, V)|Afect])):-
    findall(V1,member(v(c(X,_,_),_,V1),Afect),L), all_diff([V|L]),
    findall(V2,member(v(c(_,Y,_),_,V2),Afect),L2), all_diff([V|L2]),
    findall(V3,member(v(c(_,_,Z),_,V3),Afect),L3), all_diff([V|L3]),
    L \= L2, L3\=L, L3\=L2.

all_diff([]).
all_diff([X|Afect]):-
    \+ member(X,Afect), all_diff(Afect).
```

O nosso **estado inicial**, com o domínio referido:

```
dominio([1,2,3,4,5,6,7,8,9]).

estado_inicial(e([v(c(1,1,1),D,_),v(c(1,3,1),D,_),v(c(1,4,2),D,_),v(c(1,5,2),D,_),v(c(1,7,3),D,_),
                  v(c(2,1,1),D,_),v(c(2,2,1),D,_),v(c(2,3,1),D,_),v(c(2,5,2),D,_),v(c(2,7,3),D,_),v(c(2,8,3),D,_),v(c(2,9,3),D,_),
                  v(c(3,2,1),D,_),v(c(3,3,1),D,_),v(c(3,4,2),D,_),v(c(3,5,2),D,_),v(c(3,6,2),D,_),v(c(3,8,3),D,_),
                  v(c(4,1,4),D,_),v(c(4,2,4),D,_),v(c(4,3,4),D,_),v(c(4,4,5),D,_),v(c(4,5,5),D,_),v(c(4,7,6),D,_),v(c(4,8,6),D,_),v(c(4,9,6),D,_),
                  v(c(5,1,4),D,_),v(c(5,2,4),D,_),v(c(5,3,4),D,_),v(c(5,4,5),D,_),v(c(5,7,6),D,_),
                  v(c(6,2,4),D,_),v(c(6,3,4),D,_),v(c(6,5,5),D,_),v(c(6,6,5),D,_),v(c(6,7,6),D,_),v(c(6,8,6),D,_),v(c(6,9,6),D,_),
                  v(c(7,1,7),D,_),v(c(7,2,7),D,_),v(c(7,3,7),D,_),v(c(7,5,8),D,_),v(c(7,6,8),D,_),v(c(7,7,9),D,_),v(c(7,8,9),D,_),v(c(7,9,9),D,_),
                  v(c(8,3,7),D,_),v(c(8,4,8),D,_),v(c(8,6,8),D,_),v(c(8,7,9),D,_), v(c(8,9,9),D,_),
                  v(c(9,1,7),D,_),v(c(9,2,7),D,_),v(c(9,4,8),D,_),v(c(9,5,8),D,_), v(c(9,7,9),D,_), v(c(9,8,9),D,_),v(c(9,9,9),D,_)],


% Posições já preenchidas na tabela dada
[v(c(1,2,1),D,1),v(c(1,6,2),D,8), v(c(1,8,3),D,7), v(c(1,9,3),D,3),
v(c(2,4,2),D,5), v(c(2,6,2),D,9),
v(c(3,1,1),D,7), v(c(3,7,3),D,9), v(c(3,9,3),D,4),
v(c(4,6,5),D,4),
v(c(5,5,5),D,3), v(c(5,6,5),D,5), v(c(5,8,6),D,1), v(c(5,9,6),D,8),
v(c(6,1,4),D,8), v(c(6,4,5),D,9),
v(c(7,4,8),D,7),
v(c(8,1,7),D,2), v(c(8,2,7),D,6), v(c(8,5,8),D,4), v(c(8,8,9),D,3),
v(c(9,3,7),D,5), v(c(9,6,8),D,3)])):- dominio(D).
```

Já o **operador sucessor** é o seguinte:

```
sucessor(e([v(N,D,V)|R],E),e(R,[v(N,D,V)|E])):- member(V,D).
```

As casas já preenchidas já estão no estado inicial, marcadas com o valor respetivo.

g) Problema resolvido com algoritmo de backtracking:

```
compiling /home/pedrog/Desktop/Inteligência Artificial/Trabalhos/2º Trabalho/sudoku.pl for byte code...
/home/pedrog/Desktop/Inteligência Artificial/Trabalhos/2º Trabalho/sudoku.pl compiled, 53 lines read - 26592 bytes written, 99 ms
5 . 1 . 9 . 4 . 2 . 8 . 6 . 7 . 3
6 . 3 . 4 . 5 . 7 . 9 . 1 . 8 . 2
7 . 2 . 8 . 3 . 1 . 6 . 9 . 5 . 4
3 . 5 . 2 . 1 . 8 . 4 . 7 . 9 . 6
9 . 7 . 6 . 2 . 3 . 5 . 4 . 1 . 8
8 . 4 . 1 . 9 . 6 . 7 . 3 . 2 . 5
4 . 9 . 3 . 7 . 5 . 2 . 8 . 6 . 1
2 . 6 . 7 . 8 . 4 . 1 . 5 . 3 . 9
1 . 8 . 5 . 6 . 9 . 3 . 2 . 4 . 7
```

Demora 99 ms e escreve 26592 bytes.

h) Problema resolvido com algoritmo de backtracking com pesquisa forward:

```
compiling /home/pedrog/Desktop/Inteligência Artificial/Trabalhos/2º Trabalho/sudoku.pl for byte code...
/home/pedrog/Desktop/Inteligência Artificial/Trabalhos/2º Trabalho/sudoku.pl compiled, 53 lines read - 26592 bytes written, 100 ms

(9 ms) no
```

Por alguma razão, a nossa pesquisa forward não conseguiu chegar ao resultado final do tabuleiro (apesar de escrever uma grande quantidade de bytes, e de demorar ainda um bom tempo a correr – 100ms).

i) Para melhorar a complexidade do algoritmo, provavelmente teríamos que alterar o código principal do sudoku. Fazendo findalls de todos os valores das linhas e das tabelas pode ser dispendioso a nivel de complexidade espacial, o que depois na procura constante de entre todos os valores, vai aumentar a complexidade temporal por consequência.
Na pesquisa com forward checking, se formos tirando do domínio, por exemplo, a cada escrita na linha, aqueles números que já estão na mesma linha, irá facilitar bastante na conclusão do tabuleiro, uma vez que apenas teremos que fazer menos escolhas à medida que vamos avançando (quer nas linhas, colunas, ou quadrantes).

<u>Instruções para compilar o problema do sudoku:</u>

6. Abrir o terminal na pasta com os ficheiros
7. Abrir prolog
8. Compilar [pesquisaback] para pesquisa backtracking, ou [pesquisaforward] para pesquisa backtracking com forward checking.
9. Para além disso, dentro de ambas as pesquisas temos de fazer consult(Sudoku) (dentro do código das pesquisas).
10. Posteriormente, apenas temos que escrever “p.” no terminal e o sudoku irá ser escrito (se for encontrada solução para as nossas pesquisas.

Conclusão

Com a realização deste trabalho ficámos a ter mais conhecimento sobre a resolução de problemas como problemas de satisfação de restrições, pois aprofundámos e usámos numa vertente mais prática todos os tipos de pesquisa dados nas aulas em problemas concretos (pesquisa backtracking e pesquisa backtracking com forward checking).

Para além disso tivemos que nós próprios pensar nas melhores restrições para cada um dos problemas, o que foi bastante interessante de fazer, puxando bastante pelo nosso raciocínio.

Pensamos ter atingido a maioria dos objetivos do trabalho, principalmente para o problema do sudoku, onde chegámos à solução correta com a pesquisa com backtracking (apesar de não termos chegado com pesquisa com forward checking).

Já no problema da zebra, e percebendo bem as restrições, não chegámos a grandes valores concretos para a constituição das mesas, apenas com algum X de pessoas (provavelmente devido a alguma restrição mal implementada que não nos apercebemos).